

UTILITY FOR IDENTIFYING DIFFERENCES BETWEEN TWO JAVA OBJECTS

Reservation of Rights in Copyrighted Material

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Cross-Reference to Related Application

This patent application is related to and claims the benefit of Provisional U.S. Patent Application No. 60/467,340 filed May 5, 2003, which application is hereby incorporated herein by reference in its entirety.

Field of the Invention

This invention relates to the field of object-oriented software, and, more specifically, to a utility for differentiating between two objects and, advantageously, storing the differences for display.

Background of the Invention

Differentiating between two “things” of a similar nature is a problem frequently addressed in the area of computer science. In fact, many programmers take it for granted that a differentiation utility exists in their particular area. For example, the UNIX operating system includes a “diff” function that shows, line by line, the differences between two files. Microsoft Word includes a “Reviewing” taskbar that shows changes between and among versions of a document. In fact, many operating systems, word processors, operating environments include a procedure, function, facility, etc. that determines the differences between two items.

In more modern, object-oriented software areas, however, this differentiation utility is not always a part of the system. For example, Java only provides a way of comparing the equality of

two objects of the same Class. There is no universally defined method to determine the differences between two objects.

Therefore, a problem exists in the current object-oriented software art to differentiate between two objects of the same Class, especially in Java.

Summary of the Invention

This problem is solved and a technical advance is achieved in the art by a system and method that provides a utility for comparing two objects in an object-oriented operating system and recording the differences so that they may be put into human-readable form.

Advantageously, the objects to be compared are Java objects and the differences are recorded in XML format, which is easily transformed into human-understandable form (e.g., transform the XML into web pages using XSL).

In one exemplary embodiment of this invention, two Java objects are compared by calling one of the Java equality methods. The equality method Comparable.compareTo() is used if the objects to compare are instances of Comparable. Otherwise, the Objects.equals() method is used. If the selected equality method indicates that there is a difference between the two objects, then the get...() methods of each object are invoked in turn. The results of the get...() methods are compared. If there are differences, the differences are stored in an XML document. The get...() method is recursively invoked until the Class of the result indicates no further get...() methods to decompose.

This exemplary embodiment of this invention generates an XML document that shows the difference(s) between the two objects. To show the context of a field that is different between the two objects, the full Class hierarchy is described with attributes showing which parts are different and which parts are the same. The value of each field is displayed by a toString() call when the type of the field cannot be decomposed by further get...() calls.

Brief Description of the Drawings

A more complete understanding of this invention may be obtained from a consideration of this specification taken in conjunction with the drawings, in which:

FIG. 1 is a Warnier/Orr diagram illustrating the organization of an exemplary embodiment of this invention;

FIG. 2 is an element type diagram illustrating the XML document type declarations in accordance with the exemplary embodiment of this invention;

FIG. 3 is a sample Class definition for the purpose of illustrating operation of the exemplary embodiment of this invention;

FIG. 4 is a test case using the sample Class of FIG. 3;

FIG. 5 is the XML generated by the exemplary embodiment of this invention on the test case of FIG. 4; and

FIG.'S 6-13 comprise a program listing of an exemplary object differentiation program, called “objectDiff.”

Detailed Description

The Detailed Description section of this specification is divided into two parts. Part 1 comprises an overview of the exemplary embodiment of this invention. Part 2 comprises an example of the operation of the exemplary embodiment of this invention on a sample Class and sample data.

1. Overview of the Exemplary Embodiment of this Invention

In this illustrative embodiment of this invention, two Java objects are compared by calling one of the Java equality methods, “Comparable.compareTo()” or “Objects.equals(,” which are known in the art. While the exemplary embodiment of this invention is described using Java, one skilled in the art will appreciate that this invention can be implemented in any object-oriented language that supports method introspection after studying this specification.

According to the exemplary embodiment of this invention, the Java equality method “Comparable.compareTo()” is called if the objects to compare are instances of Comparable. Otherwise, the “Objects.equals()” method is called. If the selected equality method indicates that there is a difference between the two objects, then get...() methods of each object are invoked in turn. As will be illustrated further, below, in a get...() method, the ellipse represents the name of the method. The results of the get...() methods are compared. If there are differences between the methods, the differences are stored in an XML document. Importantly, the get...() method is recursively invoked until the Class of the result has no more get...() methods to decompose.

This exemplary embodiment of this invention generates an XML document that records the difference(s) between the two objects. To show the context of a field that is different between the two objects, the full Class hierarchy is described in XML with attributes showing which parts are different and which parts are the same. Ultimately, the value of each field is displayed by a `toString()` call when the type of the field cannot be decomposed by further get...() calls.

FIG. 1 comprises a Warnier/Orr diagram 100 illustrating the organization of a process in accordance with an exemplary embodiment of this invention. In Warnier/Orr diagram 100, four functions are defined: compare 102, createObjectDiffNode 104, compareObjects 106 and createFieldsNode 108. Each is described in turn.

Compare 102, takes two arguments, Object “a” and Object “b” (the two objects to be compared) and returns a document. Compare 102 first creates a new document 110. Document compare 102 then sets a node, “n,” to the node returned by createObjectDiffNode 104 (described below). The node “n” is then appended to the document 114 and the document is returned in 116.

The function createObjectDiffNode 104 takes three arguments: a “doc” of type Document, Object “a” and Object “b” and returns a node. At 118, CreateObjectDiffNode 104 sets a local variable “c” to the returned data from a function call to compareObjects 106 (described below). The value of “c” is checked for various conditions at bracket 120. If the

objects are different classes, then, at 122, this function returns a different class element and appends a node with the names of the two classes. If one of the objects is null, then the function returns an objectDiff element with a toString of the non-null object in 124. If both objects are null, then “null” is returned from the function in 126.

The return node “n” is set to a created objectDiff element 128 and then Object “a’s” class name element is appended to node “n” 130. In 132, if “a” is an instance of a list, then the function createListNode is called at 134, passing the arguments “doc,” “a” and “b.” If “a” is a string or an instance of a data or an instance of a number at 136, then a string element is created in 118. If “a” is anything else at 138, then the createFieldsNode 108 function is called. “n” is then returned at 140.

The function compareObjects 106 receives Object “a” and Object “b” as arguments and returns a type integer. This function first checks for exception cases in bracket 142. If, in 144, “a” and “b” are both null objects, then “0” is returned. In 146, if one of the two objects is null, then “ONE_IS_NULL” is returned. If Object “a” and Object “b” are different classes, then “DIFFERENT_CLASSES” is returned in 148.

In bracket 150, one of the Java equality methods is run on Object “a” and Object “b.” In 152, if “a” is an instance of a list, then the results of a call to the compareLists function is returned. If “a” is an instance of an object of type “Comparable” in 154, then “a” is compared to “b”. Otherwise, a.equals(b) is returned in 156.

The function createFieldsNode 108 receives “doc” of type document, “n” of type node, Object “a” and Object “b” as arguments. First, this function sets the node fields to create a “fields” element in 160. Next, in 162, this function caches the list of methods of Class “a,” whose name starts with “get” and takes no arguments and are not in the getMethodsTolgnore() list and are not “getClass” or “getThis.”

CreateFieldsNode 108 then iterates the method list 164 by invoking the method on “a” in 166, then invoking the method on “b” in 168. In 170 a field element is created with the attribute name set to the method name. In bracket 172, compareObjects 106 is called on the results of the

two method invocations 166 and 168 (above). If, in 174, the two method results are different, then an attribute is created in the document in 176 wherein the attribute is set to “true.” In 178, the function createObjectDiffNode 104 is called. Herein is the recursion in this exemplary embodiment of this invention. These functions calls to createObjectDiffNode 104 repeat until compareObjects 106 returns a value other than “TRUE.” When compareObjects 106 returns a value other than “TRUE” in 180, an attribute named “different” is set to “false” in 182 and a “toString” element is created in 184.

As stated above, an XML document is generated by the above functions that describe the differences between the two objects. The XML document type definition (DTD) is presented in table I.

```

<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT objectDiff(className, (fields I values I list))>
<!ELEMENT className (#PCDATA)>
<!ELEMENT fields (field+)> <!ELEMENT filed (objectDiff)>
<!ATTLIST field name CDATA #REQUIRED>
<!ELEMENT list (element+ I reason)>
<!ELEMENT element (objectDiff)>
<!ATTLIST element index CDATA #REQUIRED>
<!ELEMENT values (left, right)>
<!ELEMENT left (#PCDATA)>
<!ELEMENT right (#PCDATA)>
<!ELEMENT reason (#PCDATA)>
```

Table I

FIG. 2 is a relational diagram 200 of the elements of Table I. At the invocation of the differentiation function called herein “objectDiff” 202, the class name is listed 204. The fields are then listed 206. The values 208 of the fields are listed in “left” 210 and “right” 212 (corresponding to Object “a” and Object “b” above). For multiple fields 214, the name is entered 216 and “objectDiff” 202 is recursively described.

If a list 220 is developed, then the elements 222 are entered into the document along with a reason 224. An index 226 of the elements 222 is generated and “objectDiff” 202 is recursively called.

2. Example

The following three figures provide an example of the application of the above-described exemplary embodiment on sample objects of the same class. Turning first to FIG. 3, a public class named “Address” is shown as a sample class 300. This object includes five “get...()” methods: getAge() 302, getFirstName() 304, getLastName() 306, getTitle() 308 and getAddress() 310.

FIG. 4 illustrates two objects of the class Address 400. In this example, they are two simple addresses 402 and 404. FIG. 5 illustrates the XML document 500 that is generated by the exemplary embodiment of this invention. This XML document may be used by, for example, a browser, which displays the differences in human-readable form.

FIG.’S 6-13 comprise a program listing for a sample “objectDiff” function.

It is to be understood that the above-described embodiment is merely illustrative of the present invention and that many variations of the above-described embodiment can be devised by one skilled in the art without departing from the scope of the invention. It is therefore intended that such variations be included within the scope of the following claims and their equivalents.